Technická univerzita v Liberci

**Fakulta přírodovědně-humanitní a pedagogická**

**Katedra:** Ústav nových technologií a aplikované inf. (FM)

**Studijní program:** Specializace v pedagogice

**Studijní program: (kombinace)** Anglický jazyk se zaměřením na vzdělávání/Informatika se zaměřením na vzdělávání

# KURZ A NÁSTROJE PRO VÝUKU JAZYKŮ OOP

# COURSE AND TOOLS FOR TEACHING OOP LANGUAGES

# DER KURS UND MITTEL FÜR DIE UNTERRICHT DER PROGRAMMIERUNGSSPRACHEN OOP

**Bakalářská práce:** 10-FP-NTI-06

**Autor:**

Jan KONFRŠT

**Podpis:**

**Adresa:**

Strážní 916

460 14, Liberec 14

**Vedoucí práce:** Ing. Igor Kopetschke

**Konzultant:** Mgr. Jan Berki

**Počet**

| stran | slov | obrázků | tabulek | pramenů | příloh |
|---|---|---|---|---|---|
| 41 | 7366 | 5 | 1 | 18 | 0 |

V Liberci dne: 9. 12. 2010

# Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně, že jsem uvedl všechny použité zdroje a literaturu a jsem obeznámen s tím, že na mou práci se vztahuje zákon č. 121/2000 Sb. o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

**V Liberci dne 9. 12. 2010** **…….....**

**Jan Konfršt**

# Poděkování

Rád bych poděkoval Ing. Kopetschkemu, jehož rady mi byly cennou pomůckou napříč celou prací.

Stejně tak patří mé dík panu Mgr. Berkimu, který nikdy nepostrádal ochotu mě v mé snaze postrkovat správným směrem.

# Anotace

Cílem této práce je vytvořit kurz pro výuku objektově orientovaného programování uplatnitelný především na druhém stupni základní školy, případně na škole střední. Kurz byl vytvořen s přihlédnutím na možnost nulové výuky programování a jeho obsah zveřejněn ve vzdělávacím prostředí Moodle.

Klíčová slova: ZŠ, kurz, OOP, programování

# Annotation

The aim of this work was to create a course that would teach object-oriented programming to younger kids mainly at primary schools but can be applicable to high schools as well. The course was created with consideration of no programming education whatsoever and it was published on the online educational server Moodle.

Keywords: primary school, learning course, OOP, programming

# Annotation

Das Ziel dieser Arbeit ist einen Kurs für den Unterricht des objektorientierten Programmierung für die zweite Stufe der Grundschule, bzw. für die Mittelschulen zu erstellen. Dieser Kurs hat unter Bezugnahme auf die Möglichkeit der Nullunterricht enstanden und sein Inhalt wurde im Bildungsmittel Moodle veröffentlicht.

Schlüsselwörter: zweite Stufe der Grundschule, ein Kurs, OOP, das Programmierung

# Obsah

# Slovníček pojmů a zkratek

**ICT** – z anglického Information and Communication Technologies, česky Informační a komunikační technologie

**IDE** – z anglického Integrated Developement Environment, česky vývojové prostředí

**Javac** – z anglického Java Compiler – kompilátor jazyka Java

**JDK** – z anglického Java Developement Kit – vývojové nástroje jazyka Java

**JVM** – z anglického Java Virtual Machie – virtuální stroj jazyka Java, nutný pro běh programů

**JRE** – z anglického Java runtime Environment – balík obsahující knihovny nutné pro běh Java programů a virtuálního stroje

**OOP** – objektově orientované programování (nebo také z anglického Object Oriented Programming)

**RVP** – rámcový vzdělávací program, směrnice vydané MŠMT upřesňující koncepci výuky na ZŠ a SŠ v ČR

# Úvod

Cílem této práce je navrhnout možnou náplň do hodin informatiky pro základní školy. Náplň by byla uplatnitelná v podobě jednoho tématického celku v rámci bloku Informační a komunikační technologie, jenž je součástí rámcového vzdělávacího programu (dále již jen RVP).

Práce vychází z předpokladu nulové výuky programování u nás a nabízí tedy nejenom možnou alternativu k současnému stavu, ale zároveň i možnou implementaci výuky programování.

Zároveň se v této práci pokusíme zdůvodnit, že objektově orientované programování (dále již jen OOP) je vhodným typem programovacího jazyka pro počátky výuky programování.

Praktická část práce je tvořena elektronickými výukovými materiály, jež mají být dostupné na blíže nespecifikovaném portálu.

# 1 Informační a komunikační technologie

## 1.1 Vymezení pojmu

Předmětem výuky z oboru information and communication technology (dále již jen ICT) je naučit jedince zpracovávat, a tím pádem i efektivněji pracovat v naší moderní (rozumíme tím takzvaně informační) společnosti. Nutnost vzniku takového oboru je dnes dána především tím, že technologie – a výpočetní technika obzvlášť – prorůstá naší společností čím dál tím více a pro jedince je dnes již prakticky nutnost vědět, jak se v takovém světě pohybovat [1][2].

Ruku v ruce s tím, že žijeme ve věku informačním jde fakt, že život v naší společnosti je dennodenně ovlivňován jednotlivými oblastmi ICT. Často okolo sebe můžeme slýchávat větu: „Informace jsou moc," Nechceme se zde však zaobírat tím, co se tím kdo snaží říci, jak moc je to pravda, či jak je to mnohými interpretováno v dnešní společnosti, ale poukázat na takovémto prohlášení, že umět pracovat s informacemi je dnes pro fungování ve společnosti naprostá nutnost. Ostatně, dá se říci, že položka „úroveň znalosti práce s počítačem" se již pevně zakořenila v základních premisách každého kvalitního životopisu [2].

Uchopíme-li takový výrok v kontextu výuky informačně laděných předmětů na základních školách, shledáme, že výuka ICT na dnešních základních školách spočívá především v tom, že učitelé a profesoři naučí žáky základní obsluhu PC. Dále se přistupuje k základním a nezbytným úkonům, které by dnes měl zvládat již každý (kurzy pro seniory mohou být zářným příkladem, že by to měl zvládat opravdu každý), a to třeba práci s textovým či tabulkovým procesorem nebo například osvojení si dovedností vyhledávání či ověření (v ideálním případě obojí) si informací na internetu [3].

A toto všechno nám vlastně dává dohromady ucelený obrázek o práci s informacemi tak, jak by dnes výuka na ZŠ měla probíhat [3]. Viz tématické celky, které zmíníme o pár řádků níže – bez nadsázky se jedná pouze o práci s informacemi, jejich zpracování a vyhledávání.

## 1.2 Zmapování současného stavu výuky informatiky (ZŠ)

Základním kamenem hodným zamyšlení a zároveň bodem, od kterého se bude odvíjet celý zbytek této práce je výuka informatiky obecně či informaticky laděných předmětů u nás a především smysl vyučovat takové předměty. V případě této práce dokonce předměty na první pohled velmi komplexní a složité – to vzhledem k cílené věkové kategorii.

Avšak ono zmapování nebylo vlastním výzkumem jako takovým, ale bylo provedeno formou rešerše prací stávajících a dodání vlastního pohledu autora na danou problematiku. Považujeme toto řešení za pochopitelné a zdůvodnitelné, neboť pokud bychom se měli zabývat samotným výzkumem, rozsah informací, které bychom měli pokrýt, by jistě bohatě přetekl nejenom přes okraj pouze jednoho bodu této práce, ale dost možná by to bylo i na více, než jen na práci bakalářskou.

Tabulka 1 ukazuje minimální časovou dotaci v Rámcovém učebním plánu tak, jak jej vymezuje RVP pro základní vzdělávání. Vycházíme z aktuální verze, aktualizované ke dni 1. 9. 2007. Povšimněte si poměrně skromných čísel u položky Informační a komunikační technologie [3].

**Tabulka 1 – Rámcový učební plán, Rámcový vzdělávací program – základní vzdělávání [3]**

| Vzdělávací oblasti | Vzdělávací obory | 1. stupeň | 2. stupeň |
|---|---|---|---|
| | | 1. - 5. ročník | 6. - 9. ročník |
| | | Minimální časová dotace | |
| Jazyk a jazyková komunikace | Český jazyk a literatura | 35 | 15 |
| | Cizí jazyk | 9 | 12 |
| Matematika a její aplikace | | 20 | 15 |
| Informační a komunikační technologie | | 1 | 1 |
| Člověk a jeho svět | | 12 | – |
| Člověk a společnost | Dějepis | – | 11 |
| | Výchova k občanství | | |
| Člověk a příroda | Fyzika | – | 21 |
| | Chemie | – | |
| | Přírodopis | – | |
| | Zeměpis | – | |
| Umění a kultura | Hudební výchova | 12 | 10 |
| | Výtvarná výchova | | |
| Člověk a zdraví | Výchova ke zdraví | – | 10 |
| | Tělesná výchova | 10 | |
| Člověk a svět práce | | 5 | 3 |
| Průřezová témata | | P | P |
| Disponibilní časová dotace | | 14 | 24[11] |
| **Celková povinná časová dotace** | | **118** | **122** |

Jak je vidět, číslo je to velmi malé, ale na druhou stranu, současná koncepce, která stanovuje školám co a v jakém rozsahu mají učit, zároveň udává ještě takzvané disponibilní časové dotace (také viditelné v tabulce). Jedná se o hodiny, ve kterých mohou jednotlivé školy dle vlastního uvážení rozšířit stávající výuku o další vyučovací hodiny. A rozsah to není malý – 14 pro první stupeň a 24 pro stupeň druhý. Teoreticky by tedy mohly školy rozšířit stávající výuku ve všech směrech [3].

To je možností první. Druhou je, že se zde ke slovu dostávají školy, které chtějí svým žákům nabídnout něco jako specializaci. Kupříkladu škola může být zaměřená více na výtvarnou výchovu či třeba právě na informaticky laděnou výuku.

Ale disponibilní časové dotace jsou tu od toho, aby byly flexibilní, a sloužily potřebám všech předmětům či specializacím. Přirozeně tedy nemůžeme předpokládat, že by měly sloužit právě informatice. Jistě by mohly existovat případy škol, kde budou vládnout ředitelé či učitelé mající vrozený odpor vůči novým technologiím. V takovou chvíli by nám ony pevně přiřazené časové dotace pro informatiku mohly připadnout skoro až k smíchu.

Pro upřesnění, v této skromné časové dotaci mají učitelé obsáhnout následující tématické celky [3]:

- Co rozumíme pod pojmem informace. Významné informační zdroje a informační instituce. Hodnota, cena informace a informační činnosti.
- Práce s textovými informacemi.
- Vyhledávání informací.
- Práce s tabulkovými procesy.
- Přenos informací.

Pro ilustraci reality u nás se nyní odkážeme na knihu *Výzkum informační výchovy na základních školách*, která shrnuje podstatu a výsledky zajímavého (v rámci

kontextu informovanosti, připravenosti a aktivnosti českých základních škol v oblasti výuky ICT) výzkumu [1].

Publikace správně poukazuje na fakt, že ač je „moderní“ ve vyspělých školských systémech dnešní doby vyučovat informaticky laděné předměty, tak výuka není realizována tak, aby byla sjednocená a odpovídala stejným osnovám a cílům. Diference přitom vidíme v mnoha rovinách (vybíráme nejdůležitější) [1]:

- Pochopení cílů.
- Samotná deklarace programů.
- Kompetence učitelů.

Výzkum provedli doktorandi a pracovníci Katedry informačních technologií a technické výchovy Pedagogické fakulty UK v Praze v letech 2006 a 2007 (z čehož vyplývá, že výzkum byl proveden ještě před úplným zavedením RVP) za podpory MŠMT ČR. Uvědomujeme si, že tento průzkum zároveň nemůže odrážet aktuální stav, ale přes veškerou snahu jsme nebyli schopni najít aktuálnější data.

Cílem výzkumu bylo získat ucelený přehled o výuce různých předmětů v oboru ICT. Dotázáno bylo celých 3529 základních škol s výukou na druhém stupni, tj. všechny v naší republice, avšak plně odpovědělo a podaný dotazník vyplnilo „jen“ 930 z nich. To je však v publikaci vyzdvihováno jako překvapivě vysoký počet, vzhledem ke komplexnosti a rozsahu dotazníků – rovných 243 otázek. Hlavními souhrnnými poznatky tohoto výzkumu jsou [1]:

- Informační výchova neznamená jen předměty přímo s informatikou spojené, ale i předměty, kde je informatika využívána i jako pomůcka; dále řada informatických výukových aktivit, které nejsou přímo vázány na konkrétní předměty.

- Informační výchova v informačních předmětech inklinuje k výuce pouze základních aplikací a nástrojů; prohlubovací témata nejsou považována za nezbytně důležitá.
- Kompetence učitelů nejenom, že zásadně ovlivňují náplň informaticky laděných hodin, ale určují i výběr témat.
- Celá polovina učitelů přiznává tak tak dostačující kompetence pro výuku, třetina dokonce přiznává, že nemají dostatečné vědomosti (a pouze třetina respondentů má odpovídající aprobaci).
- Významným determinantem implementace ICT do vzdělávacích aktivit je časová zátěž pro učitele – projevuje se zde nepřímou úměrou zátěže a implementace ICT.
- Podmínkou efektivního rozvoje v informatických kompetencích žáků je implementace ICT do prostředí okolo nich – je lepší, když sami učitelé pro tvorbu materiálů a pomůcek používají znalostí ICT.

# 2 Kontext výuky logiky

Náplň naší práce vychází z modelové situace, kdy máme školu, hodinu informatiky a žáka s osvojenými základními dovednostmi. Míníme tím pochopitelně situaci, kdy za výchozího předpokladu pro tuto práci škola poskytuje větší časovou dotaci pro informatiku (není však podmínkou), má splněné základní body stanovené RVP a zároveň má ještě část dotace volnou. Co by mohla ještě taková základní škola nadále nabídnout svému studentovi druhého stupně či co obdobného by mohla nabídnout středoškolákovi škola střední?

Předpokládáme, že nemusíme dokládat fakt, že pokud by taková škola chtěla něco nabídnout, měla by to mít odůvodněné prokazatelným vývojem jedince a vyhnout se tak teoreticky zbytečným ztrátám času. Nakolik to tak v dnešním školství je, tím se zde nebudeme zabývat.

Ještě než ale definitivně zodpovíme, jak by takový prostor mohla daná škola naplnit, dovolíme si ukrojit malý prostor pro úvahu na téma váhy logiky ve výuce.

V roce 2004 MŠMT schválilo nové způsoby výuky žáků na prvním a druhém stupni ZŠ a na školách středních [3]. Vešly tím v platnost plány stanovující pouze tématické okruhy a tím dané kompetence žáků a studentů. Není otázkou této práce zjišťovat, jestli je to ve finále dobře a zda-li vůbec, či konkrétně jaký to mělo dopad na kvalitu výuky. Na tento krok ministerstva upozorňujeme především proto, že jedním z jeho dopadů mohlo být to, že školy nyní mají více možností, jak koncipovat a budovat studijní plány [3].

Škola má tedy volnější ruce, má možnost vyučovat žáky v těch ohledech, ve kterých má své přednosti a zároveň s tím i možnost vyhýbat se svým případným slabinám a celkově koncipovat výuku dle svého uvážení. A to vše pochopitelně za předpokladu, že dostojí základním premisám RVP [3].

Jak jsme již psali, touto prací bychom nabídli jeden další (případný) tématický celek pro výuku informatiky na ZŠ a zároveň bychom rádi poukázali na fakt, že ona výuka (OOP) by mohla mít pozitivní vliv na vývoj logiky jedince.

Snaha o to, o co se ve své práci chceme pokusit, v našem školství již do jisté míry existuje. Moc dobře si však uvědomujeme, že následující řádky nejsme schopni doložit jinak, než pouze subjektivními dojmy autora.

Můžeme se pokusit vytvořit obecné tvrzení (ale jak jsme již pravili, založené na subjektivním dojmu) o tom, že v současném školství má matematika i jinou než na první pohled praktickou funkci a že dochází k jejímu prosazování i tam, kde nemusí mít na první pohled místo.

Z našeho hlediska je tato snaha vcelku pochopitelná a odůvodnitelná, ale moc dobře si uvědomujeme, že to nemusí každý ocenit a že si mohou mnozí říci a pozastavit se nad tím, k čemu jim budou derivace, integrály či teorie strun v životě budoucím, kdy budou chtít vyučovat naše děti o pravidlech psaní měkkých nebo tvrdých i/y nebo svařovat karoserii vozu.

Zde se nemůžeme opřít o žádný konkrétní průzkum a bohužel se stejně tak nemůžeme opřít o nic podobného ani v myšlenkách, které rozvíjíme v následujících odstavcích.

Můžeme-li si dovolit hned z kraje práce polemizovat, tak ve výše zmíněném případě užití při studiu si matematika stanovuje za cíl především vést studenty k volnému rozvoji logického myšlení (a ze stejného důvodu bychom si mohli i opodstatnit snahu zavést matematiku alespoň jako povinně volitelný předmět v rámci státních maturit).

A tedy, pokud by smysl matematiky jako povinného předmětu pro většinu vysokoškolských studentů mohl být chápán jako nástroj pro podporu volného rozvoje logického myšlení studenta a snad i probouzet jiné, než čistě materiální nahlížení na problémy běžného života, tak má studium matematiky pro takového studenta snad jakéhokoli oboru rozhodně smysl. A když tvrdíme, že matematika podporuje i jiné,

než čistě materiální smýšlení, tak pro to rozhodně máme své důvody [4] – ostatně, matematika je věda plná čísel a hodnot a snad jako jediná věda je čistě abstraktní a na první pohled nenabývá konkrétních výsledků. To „tvrdil“ kupříkladu již Sokrates (ve skutečnosti se v tomto případě však jedná o smyšlený dialog významných antických postav vytvořený maďarským matematikem Alfredem Renyim) [4].

Ve zkratce řečeno – domníváme se, že matematika jistě dokáže probouzet v dostatečně otevřených jedincích možnosti nového náhledu na situace, ve kterých se mohou ocitnout. A to se pochopitelně nemusí jednat pouze o pouhou rovnici, ale prakticky o cokoliv, na co narazí v běžném životě.

A jak bychom dále mohli pomoci našemu žákovi se v oné modelové situaci (žák, počítač, hodina informatiky) se rozvíjet?

Nabídněme mu kurz programování a opět si u toho dovolme polemizovat, a to tak, že programování a logika spolu nejenom souvisí, ale že jedno na druhém závisí a také zkusíme tvrdit i to, že jedinec učící se programování se v logickém myšlení lepší. Bohužel ani zde to nemůžeme doložit konkrétními výzkumy ani daty.

Můžeme se však obrátit na článek Mgr. Berkiho, který se sice ne tak docela pozastavuje nad vyslovenou myšlenkou souvislosti programování a podpory logického myšlení, ale na druhou stranu zde autor velmi zajímavě polemizuje nad obecným smyslem výuky programování na základní škole [5]. Autor zde podtrhuje několikero základních okruhů lidského chápání, které by taková případná výuka mohla rozvinout:

- chápání struktur
- myšlení v algoritmech
- preciznost
- návyk optimalizace řešení
- znalost programovacího jazyka

Jak vidíme, programování by teoreticky smysl mohlo mít i v jiných rovinách a obecně širším záběru (ale ostatně i autor onoho článku sám podotýká, že by nerad naznačoval jasné ano či ne, neboť se v potaz musí vzít více faktorů [5]). Nicméně dílčím bodem naší hypotézy je, že co se rozvoje logiky týče a zároveň to, co pro mnohé nejen začíná, ale i končí matematikou, by v našem náhledu na problematiku mohlo nadále rozvíjet právě programování.

Programování, a konkrétně v případě naší práce OOP by v našem podání následovalo příklad matematiky a rozvedlo jej ještě do širšího měřítka. Na rozdíl od matematiky, kde se na problém nahlíží na něco, co má pevně dané řešení či alespoň konečnou množinu řešení, ke kterým se řešitel dopracovává postupným splňováním dílčích úkonů ve většinou pevně daném pořadí, nabízí programování podstatně širší náhled na zadaný problém. OOP zde totiž na rozdíl od matematiky poskytuje svému uživateli pouze sadu základních nástrojů, pomocí kterých se v teoretickém měřítku dá daná situace (nebo alespoň ve většině případů) vyřešit. A mnozí naopak dokonce ještě tvrdí [6], že OOP je mnohdy jediným možným řešením, vzhledem ke vzrůstajícím nárokům na práci a konečný výsledek programátora.

Z našeho pohledu by se dala matematika nazvat nástrojem pro řešení úloh a situací, zatímco OOP by zde bylo jen jakousi soustavou či množinou elementárních funkcí, pomocí kterých si obrazně řečeno postavíme ony podmínky a nástroje pro vyřešení zadaného problému. Problém se zde (také) stává nejenom jakousi výzvou, ale v případě OOP by se zde mohla přímo vybízet ona podpora vývoje logického myšlení jedince a mohla by jít ještě dále především díky tomu, že OOP se obecně mnohem lépe přirovnává k našemu okolnímu světu [6].

# 3 Význam OOP v kontextu práce

Dalo by se říci, že je naivní a snad i nesmyslné učit žáky, ještě k tomu tak mladé, rovnou programování objektové. Ale jak správně poukazuje Rudolf Pecinovský, který se snaží prosadit výuku OOP již na základních školách, nemá smysl učit studenty něco jiného, neboť tím akorát, ve zkratce řečeno, ztrácejí čas, možnosti nabýt pro ně levně (myšleno tím ve škole) vědomosti a vlastně si tak i krátit svoji budoucí možnou mzdu. Zároveň pan Pecinovský podotýká zajímavou věc, a to tu, že přeučovat žáky z jakéhokoli jiného typu programování je za každých okolností problém a poukazuje tedy na fakt, že nejlepší s programováním je začít právě u programování objektivního [6].

Finanční faktor jsme zmínili jen letmo, ale pravdou je, že ten je, dle Pecinovského, důležitým měřítkem a neměl by být opomíjen, rozhodně ne pod vlivem nějaké kontroverznosti, zatvrzelosti či snad ideologie co se vyučování programování týče [6].

Pecinovský zároveň udává důležité body, kterých by se měli učitelé a pedagogové držet při navrhování výuky, jsou jimi následující [6]:

- Brzy umožnit tvorbu programů.
- Nepředbíhat.
- Informace podávat postupně.
- Učit programy nejenom vytvářet, ale i ladit.
- Příklady vyžadující aktivit, musejí být zajímavé.
- Vštěpovat zásady moderního programování.
- Předkládat řešení programů nejenom triviálních.

## 3.1 Výběr jazyka

Pro práci byla jako vhodná a reprezentativní zástupkyně kategorie OOP jazyků zvolena Java. Výběr jazyka, ve kterém bychom budoucí žáky rádi školili, byl z větší části osobní volbou, neboť autor této práce má s Javou bohaté zkušenosti

a jednoduše řečeno je na ní zvyklý. Zároveň se zde ale můžeme opírat i o názory dalších.

Odvoláváme se opět na Pecinovského, který shrnul několik velmi zajímavých poznatků a požadavků na výběr OOP jazyka. Vybíráme [6]:

- Opravdová objektová orientace.
- Masivní kontrola chyb ještě před kompilací.
- Jednoduchost.
- Dostupnost na mnoha platformách.
- Bohatá nabídka knihoven.
- Pomáhat přebírat rutinní úkony – např. správa paměti.
- Nízká hardwarová náročnost.

Na první pohled Java, soudíme tak dle oficiálních definic jazyka [7][8] tomu všemu vyhovuje, ale jistě bychom se mohli ohradit, že minimálně poslední bod by se mohl považovat přinejmenším za sporný [6], a je to opět dáno vlastnostmi jazyka (princip fungování Javy a životním cyklu kódu v ní napsaném budu rozebírat dále v této práci).

I Pecinovský sám dodává, že ani Java nesplňuje veškeré požadavky jím zmíněné, ale v kombinaci s požadavky na vývojové prostředí pro něj vychází Java jako vítěz [6].

# 4 Co je to OOP

## 4.1 Definice programování, algoritmus

Rádi bychom, než se vrhneme do konkrétních detailů o rozdílech mezi objektovým a jakýmkoli jiným programovacím jazykem a ještě před samotnou definicí OOP, definovali, co to vlastně takové programování je.

Obecně řečeno, programovací jazyky slouží jako prostředky pro zápisy algoritmů, neboli krokovaných návodů potřebných pro vyřešení zadaného problému. Úspěšným zápisem takového algoritmu poté vznikne program [9][10].

A pod pojmem algoritmus si představme podrobný návod, jak krok za krokem vyřešit onen zadaný a zprvu třeba i zdánlivě neřešitelný problém. Přičemž rozdíl mezi algoritmem a již vyřešeným postupem je ten, že v rámci slova algoritmus mluvíme čistě jen o teorii, avšak v momentě, kdy jej přepíšeme do daného jazyka, již mluvíme o implementaci a tím pádem již o praktickém příkladu onoho algoritmu [10].

Programátor běžně stojí v situaci, kdy má zadaný nějaký problém a jeho úkolem není nic menšího než jej, a to pokud možno elegantně, vyřešit. Základním úkolem daného programovacího jazyka tedy především je, převést onen problém do možného řešení právě formou algoritmu. Pochopitelně v rámci programátorových možností co možná nejjednoduššího a nejefektivnějšího. Jenže takto je to až příliš jednoduše řečené a problém pro uživatele a potencionálního programátora nastává již při výběru jazyka, jelikož jak jistě mnozí tuší, není jazyk jako jazyk.

## 4.2 Dělení jazyků

Prvně dělíme programovací jazyky dle míry abstrakce, což v praxi znamená míru skrytí samotného hardwaru před programátorem. Podle tohoto faktoru dělíme ony jazyky na vyšší a nižší programovací jazyky. A pokud by se mezi případnými participanty našeho kurzu přeci jen objevil někdo, kdo již zná nějaké to

programování a setkal se již s nějakým jazykem, můžeme si být docela jistí, že se setkal s jazykem z kategorie vyšších programovacích jazyků, jelikož zájem o programovací jazyky nižší se paradoxně většinou dostavuje až ve věku vyšším, a to tehdy, když lidé začínají přemýšlet o životě, vesmíru a tak vůbec.

Vyšší programovací jazyky nám fungují jako nástroj pro komunikaci mezi uživatelem a nějakým nekonkrétním interpretrem daného jazyka. Ve vyšších jazycích se tedy staráme o algoritmus samotný a nestaráme se o to, jak jej daný stroj pochopí a jak si jej nadále bude (doslova) interpretovat, neboli lidsky řečeno chápat.

Na straně druhé máme programovací jazyky nižší, které programátorovi slouží dá se říci opačně. Nižší programovací jazyky jsou ty, kde nás již konečně zajímá samotná komunikace mezi uživatelem (dá-li se tak vůbec člověku/programátorovi, který programuje v nižším jazyce vůbec říkat) a počítačem samotným. Zde již napsané a naprogramované instrukce běžně odpovídají příkazům pro procesor samotný. Vynechává se zde tedy mezičlánek v podobě interpretujícího stroje (v případě Javy JVM), který by překládal ono naše algoritmické snažení do (pro běžné smrtelníky nečitelného) strojového kódu.

Ze samotného konceptu jazyků nižších ale logicky vyplývá jedna zásadní vlastnost, kterou by někdo mylně mohl považovat za nevýhodu. Jedná se o to, že nižší programovací jazyky se vyznačují tím, že jsou nepřenositelné, co se architektury týče, jelikož musí komunikovat již se samotným procesorem. A zde se dá předpokládat, že není procesor jako procesor a každý může chápat ony příkazy trochu jinak, respektive vůbec. Takže nižší jazyky a instrukce v nich napsané se kromě nepřenositelnosti též vyznačují tím, že jsou kompatibilní pouze s tím procesorem, pro který byly původně napsány a díky tomu se můžeme bavit o vysoké míře optimalizace.

Jenže jak jsme již naznačili, běžný programátor s rozumnými sklony k lidskosti se běžně pohybuje jen v rámci vyšších programovacích jazyků a tudíž nějaká nepřenositelnost našeho běžného uživatele-programátora vůbec netrápí. A v případě

Javy to platí hned dvakrát, jelikož Java se vyznačuje mimo jiné tím, že je velmi přenositelná, a to právě díky jejímu interpretru, JVM, který je dnes běžně dostupný na mnoha platformách. Pro programátora tedy logicky vyplývá, že mu odpadají osamělé a bezesné noci plné hraní si s Asemblerem (jazyk symbolických adres – nižší jazyk). A zároveň z toho pro něj vyplývá i to, že o optimalizaci strojového kódu se staral již někdo jiný a on se tak může věnovat své bezesné noci jen svým algoritmům.

Již jsme zmínili, proč jsme si vybral právě Javu a že jsme si ji vybral jako příklad objektově orientovaného jazyka. Ale co to vlastně znamená, co se skrývá pod pojmem „objektově orientované programování“?

Už jsme stihli nastínit, že programovací jazyky dělíme na vyšší a nižší, to dle míry abstrakce. Míra abstrakce by se dala s jistou mírou nadsázky považovat za vertikální možnost dělení programovacích jazyků a to pochopitelně značí, že opačné dělení, horizontální bude o něčem tak docela jiném [11].

Možností dalšího dělení máme mnoho, ale nastíníme již jen dělení dvojí: jazyky kompilované a interpretované a poté procedurální/neprocedurální jazyky, kde se budeme zabývat jazyky procedurálními. Ty se dále dělí na strukturované a objektové [11].

U kompilovaných a interpretovaných jazyků se rozdíl projevuje tím, jak se s kódem pracuje poté, co jej programátor obrazně řečeno vyšle do světa. U obou typů jazyka zde máme ještě jeden finální mezičlánek mezi programátorem a strojem, respektive jeho procesorem [11].

U interpretovaných jazyků zde máme „mezi-stroj“ v podobě, jak již název napovídá – interpretru. A to není nic jiného než program, který slouží k interpretaci uživatelem vytvořeného zdrojového kódu procesoru [11].

Kompilované jazyky také obsahují onen mezičlánek, ale tím je protentokrát překladač (kompilátor). Ten má za úkol napsaný zdrojový kód vzít a přeložit do strojového kódu pro daný procesor. Zde bychom se mohli pozastavit nad příkladem právě Javy, která je jedněmi považována za interpretovaný a druhými za kompilovaný jazyk [11].

Ale i přes skutečnou přítomnost Java kompilátoru (Javac) bychom Javu za kompilovaný jazyk považovat neměli. Důvodem je to, že kompilátor nám náš zdrojový kód překompiluje pouze do podoby bajt-kódu, což bychom mohli považovat teprve za mezičlánek mezi kódem zdrojovým a strojovým. Ten pak uchopí interpret jazyka (což je poslední článek mezi člověkem a strojem, tudíž bychom měli Javu opravdu považovat za jazyk interpretovaný) a interpretuje jej procesoru počítače. Interpretrem je již zmíněná JVM [7].

Dalším důležitým zmíněným typem programovacích jazyků jsou jazyky procedurální (imperativní), kam patří právě Java, potažmo OOP. Procedurální programovací jazyky jsou ty, které právě popisují úlohy jako sled funkcí a úkonů, nebo jinými slovy, oněch algoritmů. Procedurální jazyky dělíme na dva typy [11].

Prvním jsou jazyky strukturované, kde problém jako celek rozdělíme do dílčích úkonů, které řešíme de facto v řadě za sebou. Ale pozor, i když by nemusel takto zjednodušený popis tohoto postupu tak vyznít, i strukturované programování může nabízet postupy, jak řídit tok programu, rozumíme tím kupříkladu cykly či podmínky. Zde jsou však možnosti řízení toku podstatně omezenější, než je tomu u jeho vzdáleného, a, dle skromného názoru pisatele i dospělejšího, bratříčka – programování objektového.

A tím se konečně dostáváme k typu programování, o který nám šlo v první řadě – objektově orientované programování. Definovat samotné OOP není tak jednoduchý úkon, ale obecně se dá hovořit o několika základních vlastnostech a principech, které se s drobnými nuancemi opakují u všech takových jazyků a tím je tak oddělují od zbytku [12].

Základní stavební jednotkou je, jak již název promyšleně napovídá, objekt. Objektem je myšlena entita, se kterou může programátor komunikovat pomocí operací a volání a také uživatel může využít toho, že objekty si umí pamatovat svůj stav. Ona komunikace s programátorem probíhá skrz metody daného objektu. Zmíníme ještě, že programátor může vyrábět složitější konstrukce například vnořováním objektů, což je operace v OOP běžně přístupná [12].

## 4.3 Co je to objekt a jaké jsou jeho vlastnosti

Objekt si představme jako libovolný předmět z našeho reálného života, který splňuje určité vlastnosti. Komunikace s ním či s jeho funkcemi se dá provádět pouze pomocí (zvenčí) povolených volání, objekt je zapouzdřen vůči venkovnímu světu a vždy se nachází v nějakém konkrétním stavu [13].

Dalším důležitým prvkem OOP je princip zapouzdření. Jedná se o to, že objekty jsou jeden vůči druhým skryté a neukazují, co mají uvnitř. Komunikace mezi nimi pobíhá již zmíněnými voláními (s pomocí metod). Stiskneme-li na stěně vypínač, použijeme jeho metodu a pohybem ruky spustíme akci, která je nám zvenčí díky kusu plastu neviditelná a tak trochu i nepochopitelná (to pochopitelně platí za předpokladu, že nemáme maturitu ze slabo proudu či nekoukáme na vypínač rozebraný) [13].

Dalším neméně důležitým charakteristickým znakem OOP je dědičnost. O dědičnosti u objektů mluvíme v případě, kdy objekty odvozené od svých nadřazených objektů (neboli rodičů) mohou dědit vlastnosti či metody nebo naopak je zcela překrýt [13].

Ze základních vlastností OOP již zmíníme jen polymorfismus a nebudeme již dále zabíhat do dalších detailů, neboť si myslíme, že toto již stačí k tomu, aby si jedinec udělal obrázek o OOP. Polymorfismus je ta vlastnost, kdy se nám zdánlivě mohou překrývat objekty, mohou mít stejná jména, ale liší se v tom, do jaké třídy patří a tedy, odkud dědí své vlastnosti [13].

## 4.4 Java v kontextu OOP jazyků

Java je v kontextu OOP velmi zajímavým příkladem objektově orientovaného jazyka. Jelikož je takřka jediným jazykem, kde bez objektů člověk neudělá, tak říkajíc, ani ránu. Prakticky vše, co se v Javě objevuje je objektem, přičemž za slovíčkem prakticky se ukrývá pouze osm základních typů, které objekty nejsou. Už v základním příkladu všech programátorů, programu „Hello World“ vidíme pár základních principů OOP a hlavně praktickou ukázku implementace (našich) prvních objektů [13].

## 4.5 Užití jednotlivých typů jazyka

Toto je z pochopitelných důvodů velmi diskutabilní téma. Uživatel by se měl rozhodovat na základě mnoha důležitých faktorů, zmiňme třeba nabízenou funkcionalitu jazyka, neboli – obsahuje-li, a to rovnou v pokud možno pohodlné formě, to konkrétní něco (například volně dostupné knihovny s požadovanými funkcemi), s čím chceme něco vybudovat. Ruku v ruce s tímto otazníkem jde otázka druhá, co přesně chce programátor na daném jazyku vystavět a je-li tedy jím zvolený jazyk pro daný úkol vůbec vhodný. Dále máme otázku platformy, těžko bude chtít člověk kupříkladu vytvářet flashovou aplikaci na zařízení s nakousnutým ovocem ve znaku, o kterém jejich výrobce prohlašuje, že na něm Flash způsobuje chybovost a pády systému [14]. Jenže pak přichází na scénu otázka zásadní (z vlastní zkušenosti autora mající tu schopnost, že převrátí veškeré předchozí racionální úvahy naruby), a to otázka uživatelových preferencí, znalosti jazyka, a především, jeho tvrdohlavosti, rozuměj zatvrzelosti používání jednoho konkrétního jazyka na úkor jiného, mnohdy třeba mnohem užitečnějšího pro daný účel.

# 5 Výběr webového portálu

Součástí práce má být i zveřejnění výukových materiálů v rámci vhodného, ale jinak nespecifikovaného portálu.

Našimi požadavky na onen portál byl funkční a vyzkoušený systém, který by zároveň splňoval následující kritéria :

- On-line sdílení materiálů,
- zadávání cvičení, kontrola aktivity,
- umožnění žákům odevzdávat zadaná cvičení skrze tento portál.

Zvolili jsme systém Moodle, jenž má velmi vysokou penetraci v českém školství, a to hned z několika důvodů. První počeštěná verze systému Moodle spatřila světlo světa již v únoru roku 2003 a systém tak měl dostatek času se rozšířit. Dále mu pomohl fakt, že byl vyvíjen mnoha lidmi v systému ne nepodobném vývoji operačního systému Linux (časté vydávání verzí, velké množství zainteresovaných vývojářů), což vedlo ke větší oblibě systému, než pokud by byl vyvíjen stylem takzvané „katedrály“ - uzavřené společnosti a vydávání pouze odladěných a stabilních verzí, a to za cenu velkých časových prodlev a pravděpodobně i za cenu peněžitou z pohledu koncového uživatele [15].

Posledním, ale nikoliv nepodstatným důvodem, proč je Moodle dnes tak rozšířený a oblíbený i u nás je to, že jeho funkcionalita vůbec nezaostává za komerčními produkty. Za všechny jmenujme systém Blackboard [15].

# 6 Definice potřebných nástrojů pro jazyk Java

## 6.1 Bezbariérovost nástrojů a s tím spojená problematika

Dříve, než se vrhneme do jakéhokoli vypisování nástrojů, považujeme za velmi důležité podotknout pravděpodobně nejzákladnější požadavek na veškerou softwarovou výbavu, a to ten, že jednotlivé nástroje by rozhodně měly být pro studenty bezbariérové, co se jazyka týče. Rozumějme tomu tak, že veškeré nástroje by měly být v jazyce, jakém budou studenti rozumět, v našem případě tedy v češtině. Teoreticky bychom mohli zkusit přístup, kde bychom studenty tak trochu vedli k orientaci v oboru (IT) v cizím jazyce, ale zde narážíme na vcelku zásadní problém.

Problém je, že participanti našeho kurzu jsou ještě relativně mladí a můžeme tedy předpokládat, že budou ještě jazykově relativně nezkušení. Na druhou stranu se ale zde střetáváme s nutností psát a navrhovat programy tak, aby byly (z hlediska zdrojového) použitelné nejen v naší zemi, ale i za jejími hranicemi. I proto bude součástí kurzu malá zmínka o něčem jako je konvence pojmenování.

## 6.2 Výběr počítače a operačního systému

Nejzákladnějším nástrojem každého studenta by pochopitelně měl být počítač s odpovídajícím vybavením. Pod takovýmto pojmem si prvně a především představme operační systém, nejlépe takový, který bude participantovi již důvěrně známý.

Zde by se teoreticky vyplatilo udělat průzkum mezi všemi participanty kurzu, ale na druhou stranu si plně uvědomujeme, že takový průzkum by mohl být na jednu stranu zbytečný. Je nám jasné, že by se sotva kvůli takovémuto kurzu měnil operační systém. A také nutno dodat, že by nemělo sebemenší smysl participanty případně přeučovat, jelikož časová náročnost takového úkonu by velmi omezovala zbylé cíle kurzu. Ale na druhou stranu zde existuje myšlenka, že bychom měli počítače s více systémy již připravené, a ta přeci nemusí být až tak utopistická.

Vzhledem k účelu kurzu (pro vývoj je to tak obecně lepší – menší náročnost, vyšší stabilita) by bylo vhodné volit stroje postavené na předem pečlivě ozkoušené, stabilní a vhodně zvolené (například z hlediska požadavků, jmenujme třeba požadavek u Linuxu běžný – náročnost na efektivní obsluhu) distribuci Linuxu. Z vlastní zkušenosti autor může jmenovat například Debian či Ubuntu. Nicméně moc dobře si uvědomujeme, že myšlenka, že by běžný účastník kurzu byl s tímto systémem dostatečně obeznámen, je pravděpodobně utopistická. S největší pravděpodobností by tedy bylo vhodné zvolit stroje s operačním systémem Windows.

## 6.3 Vývojové balíky Javy

Dalším neméně nutným vybavením jsou potom samotné vývojové balíky Javy od Sun Microsystems obsahující potřebné nástroje pro kompilaci programů, základní soubor tříd a objektů anebo například i nějaké konkrétní ukázky programování v Javě – jedná se o Java SE Development Kit – JDK [16].

Tím to ale nekončí, studenti by měli mít k dispozici ještě balík Java Runtime Development (neobli JRE), který slouží jen jako nástroj nutný k samotnému běhu aplikací. Z teorie jen převezmeme to, že JRE obsahuje nutné knihovny pro běh a také JVM (Java Virtual Machine, je obsažena i v JDK), soukolí které nám převádí zdrojový kód do byte kódu, který už je závislý na architektuře [17].

## 6.4 Vývojové prostředí, jeho charakteristika

Následujícím nezbytným prvkem je samotné vývojové prostředí, jehož zkratka je odvozena z anglického Integrated Development Environment (dále již jen IDE). Zde by mohl, pochopitelně, posloužit i obyčejný textový editor (či nějaká jeho rozmanitější variace – kupříkladu PSPad), ale minimálně z počátku je snad pro každého jedince, který s programováním či konkrétním jazykem začíná (myšleno zcela obecně, nejedná se pouze o příklad Javy) a neví, co a jak, lepší, sáhne-li po nějakém objemnějším a komplexnějším (a především intuitivnějším a uživatelsky vstřícnějším) vývojovém prostředí. Ta dokáží většinu rutinní práce udělat za

takového nováčka a výrazně tak vyjít vstříc jeho požadavkům na jednoduchost a plynulost práce.

Vývojové prostředí (alespoň ty takzvaně „velké“) nabízejí mnoho možností, jak uživateli usnadnit práci a zefektivnit tak jeho čas strávený programováním. Zprvu podtrhněme komplexní možnosti kontroly chyb – to je něco, co ocení především nováčci. Dále nabízejí třeba intuitivní doplňování uživatelem načatých slov (podobně, jako to dnes dělávají pokročilejší textové procesory) či podobně fungující doplňování předpokládaného užití proměnných či volání/metod. Další velikou a velmi užitečnou výhodou jsou nástroje a funkce pro zpřehlednění samotného kódu, mluvíme tu o odrážkách a odsazeních v základu, a pak třeba i takové „detaily“ jako je barvené odlišení jednotlivých prvků (metody, proměnné, hodnoty, …). Avšak základním pomocníkem, kterého snad každý začátečník s vděkem využije jsou nástroje pro kompilaci programů integrované přímo do IDE, takže uživatel ve finále jen obrazně řečeno stiskne spoušť a má zkompilováno.

V rámci oněch „velkých“ a nekomerčních IDE bychom měli na výběr především mezi dvěma nejrozšířenějšími programy, a to Eclipse a Netbeans. Zajímavostí je, že druhý jmenovaný pochází původně z Česka a nyní je vyvíjen pod hlavičkou samotného Sun Microsystems, respektive společností Oracle, která před nedávnem Sun Microsystems koupila. Někdy ze zcela neznámých důvodů bývá zvykem, že se vedou nesmyslné žabomyší války o tom, které prostředí je lepší, ale výsledkem zůstává, že ve finále se jedná o rozhodnutí víceméně dle vkusu a osobních preferencí. Sluší se ale podotknout, že jednotlivé výhody a nevýhody konkrétních vývojových prostředí ztrácejí tak docela pro absolutního nováčka smysl – nemá šanci rozpoznat a identifikovat případné chyby či nedostatky a stejně tak ještě zdaleka nebude schopen ocenit všechny výhody.

Tato velká IDE jsou však mnohými považovány za žraloky [6], a to především kvůli nárokům na hardware počítače. Pro náš kurz jsme zvolili vývojové prostředí podstatně kompaktnější a dokonce původně navržené pro výuku OOP – BlueJ[6]. Je kompaktní, jednoduché, intuitivní a v neposlední řadě i počeštěné (Pecinovský na

svých osobních stránkách nabízí ke stažení svojí verzi českého překladu, kterou doporučujeme pro použití s kurzem).

Na straně druhé zde ale existují i komerční alternativy. Zmiňme například IntelliJ IDEA od JetBrains a IBM Rational Application Developer. Druhý zmiňovaný, jak už název dává matně tušit, je vyvíjen pod hlavičkou jedné z nejstarších a nejzkušenějších firem v oboru IT vůbec.

# 7 Definice oblastí výuky

## 7.1 Výstupní kompetence participantů a obsah výuky

Naším prvním rozumným krokem k naplnění praktické části této práce by pochopitelně mělo být definování toho, co bychom chtěli účastníky kurzu vlastně naučit. Zprvu bychom rádi zdůraznili, že rozhodně není naším cílem vytvořit pro participanty nějaké objektově orientované síto, ze kterého by nám pak vzešla hrstka použitelných programátorů, ostatně to by v této věkové kategorii pravděpodobně ani nebylo možné. Cílem je poskytnout co možná nejširší skupině lidí (již jsme zmínili, že předchozí znalost programování není podmínkou) solidní základ, na kterém si pak mohou sami zapracovat.

Za úkol jsme si tedy dali ukázat participantům, jaký má OOP (potažmo Java) potenciál a čeho všeho se s ním/ní má dosáhnout. Žáci si zkusí i programování vlastní, ale nebudeme rozebírat věci typu řízení toku programu či třeba vlastnosti jednotlivých proměnných. Náš kurz pojednává především o vlastnostech OOP jako takového a jak se s nimi zachází k užitku programátora.

A co tedy vlastně z Javy žáci uvidí? Jak jsme již naznačili, výuka se bude motat především okolo vlastností OOP jazyka a jednotlivé okruhy a cvičení budou koncipovány tak, aby tyto vlastnosti pokrývaly. Vše si budou moci žáci osobně vyzkoušet v prostředí BlueJ.

## 7.2 Zpětná vazba

Jedním z našich požadavků na webový portál byl ten, že žáci budou moci odevzdávat s jeho pomocí cvičení. Cvičné úlohy jsou navrženy tak, aby probíhaly v hodině – žáci zároveň s výkladem látky dostávají úkoly a ty mají následně za pomoci portálu (Moodle) odevzdat. Odevzdávat budou projekty, které budou postupně vytvářet ve vývojovém prostředí, které jim předem připravíme (BlueJ).

# 8 Konkrétní studijní plány pro výuku jazyka.

## 8.1 Souhrn kurzu, časová náročnost, kompetence

V rámci omezené doby trvání (návrhem je jeden měsíc – 3 týdny, dvě vyučovací hodiny v bloku/týden) není možné, aby poté z kurzu odešel jedinec kompetentní cokoliv většího sám vytvářet, ostatně, jeho kompetencí nebude nic jiného než získat vhled do OOP jako takového a víceméně tedy spíše jen dostat možnost vidět, co a jak funguje a rozhodnout se, zda-li by se tomu chtěli případně věnovat dál, či zkusit jiný typ programování nebo myšlenku stát se programátorem vypustit úplně.

## 8.2 Forma

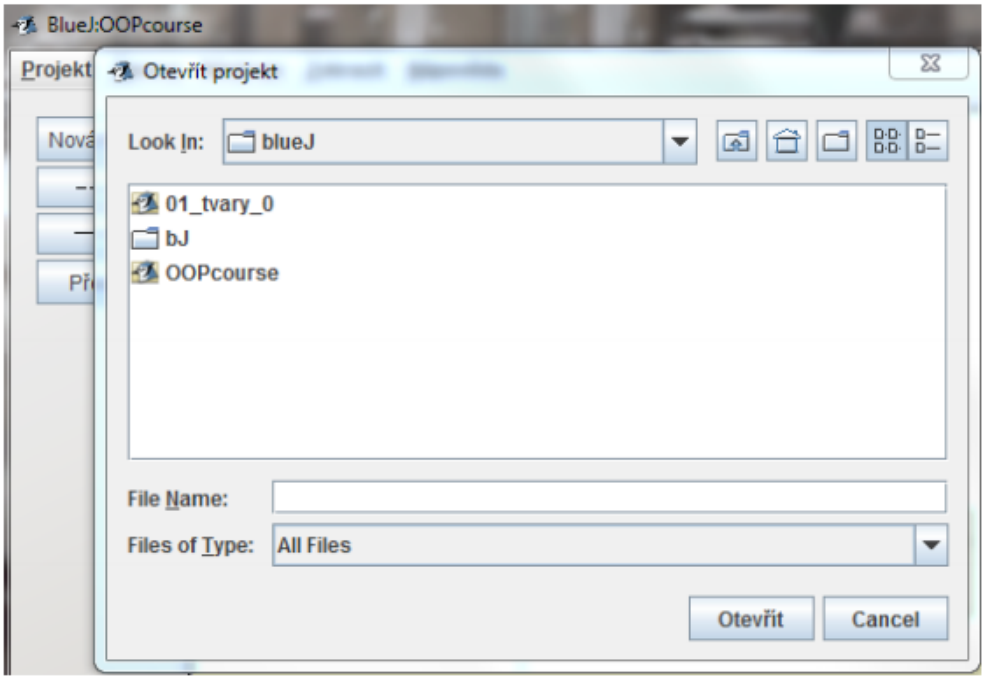
Veškeré materiály jsou přístupné v elektronické podobě na portálu Moodle (FPHP) na adrese https://moodle.fp.tul.cz/course/view.php?id=1074 (viditelné po přihlášení na portál, kurz je chráněn heslem („oop_kurz“), ve formě .pdf dokumentů). Zde v práci jen pro úplnost uvedeme formou obrázku krátký náhled na konečnou podobu výukových materiálů.

Při tvorbě konkrétních studijních materiálů jsme vystupovali především z materiálů a podkladů Pecinovského, jehož sice nikde konkrétně necitujeme, ale držíme se (ne však naprosto přesně) kostry jím koncipované výuky [6][18].

Považujeme za důležité zmínit, že materiály jsou určené jako podklady pro učitele, je k nim nutný výklad a nejsou vhodné pro samostudium.

# 1 Začínáme s OOP

Začneme tím, že si otevřete program BlueJ a z kurzu na Moodlu si stáhnete projekt, který si následně v BlueJ otevřete. Klepněte na **Projekt → Otevřít projekt...** a v následující dialogovém okně si najdete svůj stažený projekt. Měl by mít ikonku jako projekty na obr. 1.




**Obr. 1 – výběr projektu**

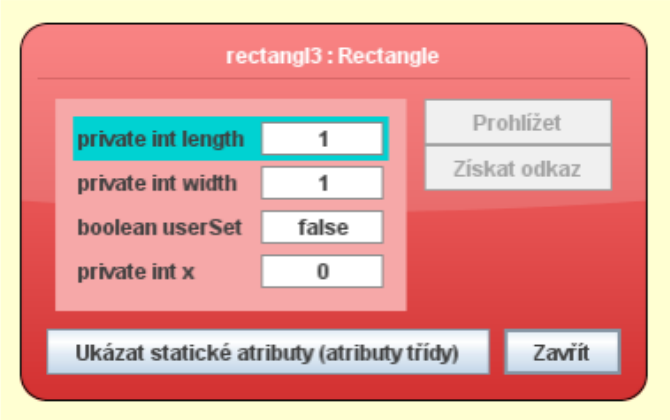
## 1.1 Objektově orientované programování

Objektově orientované programování je jedním z mnoha typů, jak se dá k programování přistupovat. Vychází ze základní myšlenky, že programátoři v něm vytváří a modeluje svět tak, že vlastně **reprezentuje okolní svět a vztahy v něm.**

Objektem se myslí konkrétní realizace nějaké třídy, neboli její instance. Porozhlédnete-li se po místnosti, máte pravděpodobně možnost spatřit na jedné nebo více stěnách vypínač. To je **instance** třídy vypínačů – jeden její konkrétní zhmotnělý příklad → **objekt**.

**Obr. 1 - ukázka kurzu – úvod kurzu**

Ten červený obdélník nám v programu BlueJ funguje zároveň jako interaktivní knoflík zprostředkovávající komunikaci s objektem. Klepneme-li na něj pravým tlačítkem myši rozbalí se nám kontextová nabídka nabízející několikero možností (zpráv neboli metod). Ty se nacházejí v menu nahoře, dole (pod čarou) máme možnosti, které nám nabízí sám BlueJ. Klepněme si na **Prohlížet** a podívejme se, co se stalo: (obr. 6)




**Obr. 6 – vlastnosti objektu**

Zde vidíme základní vlastnosti objektu **rectangl3** jenž je instancí třídy **Rectangle**. Máme zde délku (**length**), šířku (**width**), jednu logickou proměnou **userSet** (logická proměnná může nabývat pouze hodnot **true** a **false**, neboli pravda/nepravda).

**Obr. 2 – ukázka kurzu – obrázek z vývojového prostředí**

## 1.5 Zasílání zpráv s parametry

Zasílání zpráv (neboli volání metod) s parametry není vůbec nijak složité. V našem prostředí BlueJ to je velice podobné vytváření instancí s parametry.

**Úkol 4**

Pokud nemáte, vytvořte si instanci třídy **Rectangle** a poté zavolejte metody **setLength(int value)** a **setWidth(int value)**.

Program se vás poté zeptá na celočíselné hodnoty typu int. Můžete také zkusit zadat hodnoty s desetinnou čárkou a sledovat, co se poté stane. Máte pro to vysvětlení?

## 1.6 Zprávy vracející hodnoty

Objekty mohou na zasílání zpráv reagovat různě, v předchozím případě jsme viděli reakci typu „nastav něco“, další důležitou kategorií jsou zprávy vracející hodnoty a dají se tedy popsat asi jako: „vrať něco“.

**Úkol 5**

Zašlete zprávu **getLength()** zinicializovanému objektu typu **Rectangle**. Co se stalo?

Takové zprávy nám mohou vracet nejenom libovolné datové typy, ale ve složitějších případech mohou také dokonce vracet objekt samotný.

**Obr. 3 – ukázka kurzu – zadávání cvičení v hodině**

Jednoduše řečeno, obě metody se jmenují stejně, dejme tomu, že jste si je pojmenovali následovně:

```
setValue(int parameter)
{
    a = parameter;
}

setValue(String parameter)
{
    str = parameter;
}
```

**Obr. 4 – ukázka kurzu – psaní zdrojového kódu**


**Obr. 5 – ukázka kurzu – kurz implementovaný do systému Moodle**

# 9 Závěr

Cílem práce tedy bylo navrhnout kurz, který by se dal použít na základní škole jako pomůcka při výuce konkrétního tématu v rámci informatiky – programování.

Při mapování současného stavu jsme shledali, celkově je výuka informatiky problém, jelikož v českém školství není dostatek kompetentních učitelů. Dále zde existuje reálná možnost, že dnes je pravděpodobnější, že pokud by chtěla škola zakomponovat výuku programování do svých plánů, sáhne v dnešní době spíše po programování jiného typu. K zamyšlení rovněž stojí u nás daná (malá) časová dotace pro informaticky laděné předměty.

Samotné výukové materiály jsme pojali pouze jako vhled do OOP samotného, nikoli podrobnější výzkum vlastností a funkcí Javy. Tyto materiály jsme zároveň postavili tak, že mají sloužit především jako pomůcka učitelům a je k nim ještě třeba jejich mluveného slova. Nehodily by se tedy, pokud by kurz měl probíhat jen formou samostudia.

V materiálech jsme se snažili vyhnout typu výuky často viděné v učebnicích, kdy se žáci učí spíše samotný jazyk než principy (OOP).

Zásady pro vypracování jsme splnili, na žádný neřešitelný problém nenarazili. A i přesto, že autor sám nemá žádné zkušenosti s takovouto výukou v praxi, tak jsme přesvědčeni, že práce tímto obsahuje vše potřebné pro případnou aplikaci kurzu a je použitelná v praxi.

Úplným závěrem ještě dodáme, že aplikace tohoto kurzu by byla možná i na škole střední, a to bez větších zásahů do samotného obsahu. Záleželo by však na tom, jestli by na SŠ nemohla časová dotace být větší. V takovém případě by bylo záhodno kurz ještě rozšířit.

# Rejstřík použité literatury

**zdroje uvedeny v pořadí, v jakém se objevily v textu**

[1] RAMBOUSEK, Vladimír a kol. Výzkum informační výchovy na základních školách. Plzeň, Koniáš, 2007. ISBN: 80-86948-10-2

[2] ČESKÝ TRH PRÁCE s.r.o. *Práce v ČR – Jak správně napsat strukturovaný životopis – CV ?* [online]. [cit. 13. 8. 2010]. URL: <http://www.pracevcr.cz/?idl=38&pomid=141>

[3] Přispěvatelé RVP. *Metodický portál RVP* [online]. [citováno 23. 8. 2010]. URL: <http://rvp.cz/>

[4] RÉNYI, Alfred. Dialogy o matematice. Praha, Mladá Fronta, 1980. Český překlad: Jan Králík. ISBN: 23-110-80

[5] BERKI, Jan. *Programování povinně v kurikulu?* [online]. Publikováno: 5. 11. 2009 [citováno: 25. 11. 2010]. URL: <http://clanky.rvp.cz/clanek/c/Z/6211/programovani-povinne-v-kurikulu-.html/>

[6] PECINOVSKÝ, Rudolf. *Proč a jak učit OOP žáky základních a středních škol* [online]. Publikováno: 1. 10. 2004, [cit. 5. 10. 2010]. URL: <http://vyuka.pecinovsky.cz/prispevky/Proc_a_jak_ucit_OOP_na_ZS_a_SS.pdf>

[7] Přispěvatelé Wikipedie. *Java (programming language). Wikipedia, the free encyklopedia* [online]. Poslední revize 24. 5. 2010 [cit. 27. 5. 2010] URL: <http://en.wikipedia.org/wiki/Java_(programming_language)>

[8] Příspěvatelé Oracle. *Oracle and Java. Technologies* [online]. [cit. 13. 9. 2010] URL: <http://www.oracle.com/us/technologies/java/index.html>

[9] Přispěvatelé Wikipedie. *Programming language. Wikipedia, the free online encyclopedia* [online]. Poslední revize 18. 5. 2010 [cit. 27. 5. 2010] URL: <http://en.wikipedia.org/wiki/Programming_language>

[10] Přispěvatelé Wikipedie. *Algorithm. Wikipedia, the free online encyclopedia* [online]. Poslední revize 25. 5. 2010 [cit. 27. 5. 2010] URL: <http://en.wikipedia.org/wiki/Algorithm>

[11] Přispěvatelé Wikipedie. *Programovací jazyk. Wikipedie* [online]. Poslední revize 24. 5. 2010 [cit. 27. 5. 2010] URL: <http://cs.wikipedia.org/wiki/Programovac%C3%AD_jazyk>

[12] Přispěvatelé Wikipedie. *Object-oriented programming. Wikipedia, the free online encyclopedia* [online]. Poslední revize 26. 5. 2010 [cit. 27. 5. 2010] URL: <http://en.wikipedia.org/wiki/Object-oriented_programming>

[13] SHARON Zakhour a kol. JAVA 6 výukový kurz. Brno, Computer Press, 2007. ISBN: 978-80-251-1575-6

[14] MICK Jason. *DailyTech – Apple's Jobs Says Flash Crashes Macs; No Flash for iPad, iPhone Planned* [online]. Publikováno 19. 2. 2010 [cit. 30. 9. 2010] URL: <http://www.dailytech.com/Apples+Jobs+Says+Flash+Crashes+Macs+No+Flash+for+iPad+iPhone+Planned/article17738.htm>

[15] MUDRÁK David. *Implementace vzdělávacího systému Moodle v českých školách* [online]. Publikováno 23. 6. 2005, poslední revize 10. 11. 2005 [cit. 25. 11. 2010] URL: <http://www.lf1.cuni.cz/Data/files/E-learning/moodle.pdf>

[16] Přispěvatelé Oracle. *READ ME – Java Standart Platform, Standart Edition Developement Kit* [online]. [cit. 20. 8. 2010] URL: <http://www.oracle.com/technetwork/java/javase/readme-142177.html>

[17] Přispěvatelé Oracle. *Java Platform, Standart Edition Runtime Environment README* [online]. [cit. 20. 8. 2010] URL: <http://www.oracle.com/technetwork/java/javase/jrereadme-182762.html>

[18] PECINOVSKÝ Rudolf. *Výuka objektově orientovaného programování* [online]. Poslední revize 12. 8. 2010 [cit. 6. 10. 2010] URL: <http://vyuka.pecinovsky.cz/>